

Genesis: An Extensible Java

by

Ian Lewis BComp Hons

Submitted in fulfilment of the requirements for the

Degree of Doctor of Philosophy

University of Tasmania

February 2005

This thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information and duly acknowledged in the thesis, and to the best of the candidate's knowledge and belief no material previously published or written by another person except where due acknowledgement is made in the text of the thesis.

Ian Lewis

This thesis may be made available for loan and limited copying in accordance with the
Copyright Act 1968.

Ian Lewis

Abstract

Extensible programming languages allow users to create fundamentally new syntax and translate this syntax into language primitives. The concept of compile-time meta-programming has been around for decades, but systems that provide such abilities generally disallow the creation of new syntactic forms, or have heavy restrictions on how, or where, this may be done.

Genesis is an extension to Java that supports compile-time meta-programming by allowing users to create their own arbitrary syntax. This is achieved through macros that operate on a mix of both concrete and abstract syntax, and produce abstract syntax. Genesis attempts to provide a minimal design whilst maintaining, and extending, the expressive power of other similar macro systems.

The core Genesis language definition lacks many of the desirable features found in other systems, such as quasi-quote, hygiene, and static expression-type dispatch, but is expressive enough to define these as syntax extensions. User-defined macros produce only well-formed syntactic structures via the use of a predefined set of classes that define a Java abstract syntax.

At the heart of Genesis is a flexible parser that is capable of parsing any context-free grammars — even ambiguous ones. The parser is capable of arbitrary speculation and will consider all possible parses. The parser constructs a graph of possible paths, and is capable of dynamically pruning this graph, or combining nodes, based on precedence or associativity rules. This general parser allows macro programmers to forget about parsing, and concentrate on defining new syntax.

One key goal of this system was to address the programmer's learning curve by providing as simple a system as possible. This was achieved by the use of the flexible parser, the introduction of only one new construct to standard Java, and extensions to make programming macros more user friendly.

The expressiveness of Genesis is wide ranging; it is capable of providing small scale limited use macros, large scale semantic modifications, through to complete language replacements.

To demonstrate this expressiveness, we implement many of the simple test cases found in other systems, such as a type-safe `printf`, assertions, and iteration statements. These test cases require an ability to perform static type-checking and to manipulate compile-time values and abstract syntax trees. As additional examples of Genesis' expressive power we also provide implementations of embedded subsets of SQL and Haskell. As a final proof of power, the Haskell subset can operate as a stand-alone extension independent of any recognisable Java code.

Acknowledgements

Thanks to Vishv for being my most long-suffering supervisor and for steadfastly maintaining to this day that he has no idea what I am doing whatsoever.

Also, to Arthur and Julian for helping get me over the line, battered and bruised though I may be.

Thanks to the School of Computing for providing support when it was really necessary and without which I would never even come close to finishing.

To Julian for showing me no matter how annoyingly pedantic I may have thought myself there is always someone worse, and who will scare me.

To Nicole simply for torturing countless cats (as is my understanding) rather than slaying me alive simply because, as my office neighbour, she was Ned to my Homer.

To Adam for giving us all someone to mutter incessantly under our breaths about at his appearance of progress and for all the tennis ball humour — just hilarious. Oh yeah, and for proof-reading.

Thanks to Mark for all of those closed door discussions, the whiteboard Jeffisms, misogynistic research proposals, and for becoming a living urban legend.

To Byeong for showing me that it really is worth taking the time to greet people, and for being brave enough to come drinking with me — once.

To Craig, for scheduling his holiday for the worst possible time... and for making up for it by scheduling another celebratory one once this ordeal is finally over.

Thanks to Banjo's, Hungry Jacks, Eastlands, Centrepont, Northgate, KFC, and their ilk for donating places where, strangely, I initially found myself most productive.

Particularly thanks to those who donated their power to make this all possible.

To my darling girlfriend Kim for tolerating all those assurances of “two more weeks”, and just being tolerant in general.

And finalee, tanks to those harmleff alcamahol molecools that didn't hampa me won bit.

Table of Contents

ABSTRACT	VII
ACKNOWLEDGEMENTS	IX
TABLE OF CONTENTS	XI
LIST OF CODE EXAMPLES	XVIII
LIST OF FIGURES	XXI
LIST OF TABLES	XXII
1 INTRODUCTION	1
1.1 OVERVIEW	2
1.1.1 Assessment of Success	3
1.1.2 Conventions	4
1.2 SOURCES OF CODE EXAMPLES	6
1.2.1 Java	6
1.2.1.1 <i>Java1.4</i>	6
1.2.1.2 <i>Java1.5</i>	7
1.2.2 MultiJava	7
1.2.3 Maya	7
1.2.4 Java Syntax Extender	8
1.2.5 OpenJava	8
1.2.6 Haskell	8
1.2.7 Template Haskell	9
1.2.8 Lisp	9
1.2.8.1 <i>Common Lisp</i>	9
1.2.8.2 <i>Scheme</i>	9
1.2.9 C++	10
1.2.10 MS ²	10
1.2.11 SQL	10
1.2.12 Pro*C	10
1.2.13 Icon	11
1.2.14 Ada	11
1.3 THESIS STRUCTURE	12
2 EXTENSIBILITY	15
2.1 OVERVIEW	16
2.2 DEFINING EXTENSIBLE	17
2.2.1 Other Similar Terminology	17
2.2.2 Previous Extensibility Definitions	17
2.2.3 Definition of Extensible	18
2.3 LANGUAGE EXTENSION MECHANISMS	20
2.3.1 Library Systems	20
2.3.1.1 <i>Threading</i>	21
2.3.1.2 <i>SQL</i>	22
2.3.1.3 <i>Summary</i>	24
2.3.2 Open Compilers	24
2.3.2.1 <i>Glasgow Haskell Compiler</i>	25
2.3.2.2 <i>Summary</i>	25
2.3.3 Text Macros	25
2.3.3.1 <i>Assertions</i>	27
2.3.3.2 <i>Iteration</i>	28
2.3.3.3 <i>Generators</i>	28
2.3.3.4 <i>Message Maps</i>	33
2.3.3.5 <i>DEBUG_NEW</i>	34
2.3.3.6 <i>Summary</i>	35
2.3.4 Two-tier Languages	35
2.3.4.1 <i>C++ Template Meta-programming</i>	36
2.3.5 Integrated Language Features	39

2.4	IS EXTENSIBILITY NECESSARY?	40
2.4.1	What is Necessary?	42
2.4.2	Language Modifications	43
2.4.2.1	<i>Language Standards Revision / Development</i>	44
2.4.2.2	<i>Research Languages</i>	44
2.4.2.3	<i>Embedded Languages</i>	45
2.4.2.4	<i>Java1.5</i>	46
2.4.3	How Much Power is Too Much Power?	47
2.4.4	Multi-paradigm languages	48
2.4.5	Previous Interest	49
3	ASSESSMENT OF EXTENSIBILITY	51
3.1	OVERVIEW	52
3.2	DESIRABLE LANGUAGE PROPERTIES	53
3.2.1	Hygiene and Referential Transparency	54
3.3	CRITERIA FOR RATING EXTENSIBLE LANGUAGES	56
3.3.1	Power	56
3.3.2	Usability	57
3.3.3	Error Handling	58
3.3.4	Previous Research	58
3.4	BENCHMARK TEST CASES	60
3.4.1	Simple constructs	61
3.4.1.1	<i>Assertions</i>	61
3.4.1.2	<i>Iteration</i>	63
3.4.1.3	<i>Typesafe Formatted Output</i>	65
3.4.2	Complex Constructs	67
3.4.2.1	<i>SQL Subset</i>	67
3.4.2.2	<i>Generators</i>	68
3.4.2.3	<i>Haskell Subset</i>	69
4	REVIEW OF EXTENSIBLE LANGUAGES	75
4.1	OVERVIEW	76
4.2	LISP / SCHEME	78
4.2.1	Power	79
4.2.1.1	<i>S-expressions</i>	79
4.2.1.2	<i>Macros: defmacro</i>	80
4.2.1.3	<i>Macros: define-syntax</i>	81
4.2.1.4	<i>Syntax Creation</i>	82
4.2.2	Usability	82
4.2.2.1	<i>Name Capture: gensym</i>	83
4.2.2.2	<i>Name Capture: define-syntax Revisited</i>	83
4.2.3	Error Handling	84
4.2.4	Applicability to Benchmarks	85
4.2.5	Extensibility Criteria Assessment	86
4.3	TEMPLATE HASKELL	88
4.3.1	Power	88
4.3.1.1	<i>Reification</i>	89
4.3.2	Usability	89
4.3.2.1	<i>Splicing</i>	90
4.3.2.2	<i>Quasi-quotation</i>	91
4.3.3	Error Handling	91
4.3.4	Worked Examples	92
4.3.4.1	<i>Type-safe Formatted Output</i>	92
4.3.4.2	<i>Selection From an N-tuple</i>	93
4.3.5	Applicability to Benchmarks	94
4.3.6	Extensibility Criteria Assessment	95
4.4	META SYNTACTIC MACRO SYSTEM	97
4.4.1	Power	97
4.4.2	Usability	97
4.4.3	Error Handling	98

4.4.4	Worked Examples	98
4.4.4.1	<i>Dynamic Binding</i>	98
4.4.4.2	<i>Extended Enumerations</i>	100
4.4.5	Applicability to Benchmarks	102
4.4.6	Extensibility Criteria Assessment	103
4.5	JAKARTA TOOL SUITE	105
4.6	JAVA SYNTAX EXTENDER	106
4.6.1	Power	106
4.6.2	Usability	107
4.6.3	Error Handling	108
4.6.4	A Worked Example: foreach	108
4.6.4.1	<i>Underlying Implementation</i>	108
4.6.4.2	<i>The syntax Macro</i>	109
4.6.5	Applicability to Benchmarks	110
4.6.6	Extensibility Criteria Assessment	111
4.7	OPENJAVA	113
4.7.1	Power	113
4.7.2	Usability	114
4.7.3	Error Handling	114
4.7.4	Applicability to Benchmarks	115
4.7.5	Extensibility Criteria Assessment	115
4.8	MAYA	117
4.8.1	Power	117
4.8.1.1	<i>Grammar Productions and Semantic Actions</i>	117
4.8.1.2	<i>Laziness</i>	117
4.8.1.3	<i>Overloading Mayans</i>	118
4.8.2	Usability	118
4.8.3	Error Handling	119
4.8.4	Worked Examples	119
4.8.4.1	<i>Assertions</i>	119
4.8.4.2	<i>Iteration</i>	120
4.8.5	Applicability to Benchmarks	123
4.8.6	Extensibility Criteria Assessment	124
5	GENESIS LANGUAGE DEFINITION	127
5.1	OVERVIEW	128
5.2	DESIGN RATIONALE	129
5.2.1	Arbitrary Syntax Creation	129
5.2.2	Compile-time Interrogation	129
5.2.3	Base Language	130
5.2.4	Outward Language Simplicity	130
5.2.4.1	<i>Programmer Support</i>	131
5.2.4.2	<i>Parser Restrictions</i>	131
5.2.5	Inward Language Simplicity	131
5.2.6	Error Reporting	132
5.3	MACRO DEFINITIONS	133
5.3.1	Parameters	133
5.3.2	Precedence	134
5.3.2.1	<i>Precedence by Grammar Modification</i>	135
5.3.3	Associativity	136
5.3.3.1	<i>Associativity by Grammar Modification</i>	136
5.3.4	Zero Argument Macros	136
5.3.5	Modifiers	137
5.3.6	Exceptions	137
5.3.7	Macro Body	138
5.3.8	Evaluation Order	138
5.3.9	Placement and Scope	138
5.3.10	Grammar	139
5.4	TOKENISING	140
5.4.1	Tokenising Approach Overview	140

5.4.2	Special Cases	141
5.4.3	Symbol Handling	141
5.4.3.1	Traditional Approach	142
5.4.3.2	Explicit Spaces	142
5.4.3.3	Single-character Symbols	142
5.4.3.4	Symbol Combinations	143
5.4.4	Tokeniser Algorithm	144
5.5	MACRO EXPANSION	146
5.5.1	Import Mechanism	146
5.5.2	Expansion Strategy	146
5.5.2.1	Evaluation Order	146
5.5.2.2	Construction Versus Manipulation	147
5.5.2.3	Outermost Versus Innermost Evaluation	148
5.5.2.4	Non-destructive Restriction	149
5.5.2.5	Standard Usage	149
5.5.3	Macro Matching	150
5.5.4	Precedence	150
5.5.4.1	Same Sub-rules	151
5.5.4.2	Different Sub-rules	151
5.5.5	Associativity	152
5.5.6	Exceptions	153
5.5.7	Restrictions	153
5.6	STANDARD ENVIRONMENT	155
5.6.1	Abstract Syntax Classes	155
5.6.2	Exceptions	157
5.6.3	Compile-time Typing	157
5.6.4	Macro Reflection	158
6	REVIEW OF PARSERS	159
6.1	OVERVIEW	160
6.1.1	Extensible Language Parsing	160
6.1.1.1	Usability	161
6.1.1.2	Mid-parse Grammar Modification	161
6.2	GRAMMARS	163
6.2.1	Grammar Structure	163
6.2.2	Chomsky Hierarchy	163
6.2.3	Context-free Grammars	164
6.2.3.1	Backus-Naur Form	165
6.2.3.2	Extended BNF	165
6.2.4	Grammar Properties	166
6.2.4.1	Ambiguous Grammars	166
6.2.4.2	Left- and Right-recursive Grammars	167
6.3	PARSERS	168
6.3.1	Derivation	168
6.3.2	Naming	169
6.3.2.1	Derivation Categorisation	169
6.3.2.2	Lookahead Categorisation	169
6.3.3	Tokenising	170
6.3.4	Parsing Methods	171
6.4	TOP-DOWN PARSING	172
6.4.1	Predictive Parsing	172
6.4.1.1	Advantages	173
6.4.1.2	Disadvantages	173
6.4.2	Parse Table Approach	174
6.4.2.1	Advantages	176
6.4.2.2	Disadvantages	177
6.4.3	Suitability to Extensible Languages	177
6.4.3.1	Usability	177
6.4.3.2	Mid-parse Grammar Modification	177
6.5	BOTTOM-UP PARSING	178

6.5.1	Shift-reduce Parsing	178
6.5.2	Operator-precedence Parsing.....	179
6.5.2.1	Advantages.....	181
6.5.2.2	Disadvantages.....	181
6.5.3	LR Parsing.....	182
6.5.3.1	LALR Parsing	185
6.5.3.2	SLR Parsing.....	187
6.5.3.3	Advantages.....	187
6.5.3.4	Disadvantages.....	188
6.5.4	Suitability to Extensible Languages	188
6.5.4.1	Usability.....	188
6.5.4.2	Mid-parse Grammar Modification	189
6.6	GENERAL PARSING	190
6.6.1	CYK Parsing	190
6.6.2	Earley's Algorithm.....	191
6.6.3	Chart Parsers	192
6.6.4	Suitability to Extensible Languages	193
6.6.4.1	Usability.....	193
6.6.4.2	Mid-parse Grammar Modification	193
6.7	ANALYSIS	194
6.7.1	Power.....	194
6.7.2	Efficiency	195
6.7.3	Suitability to Extensible Languages	195
6.7.3.1	Usability.....	195
6.7.3.2	Mid-parse Grammar Modification	196
7	GRAPH EXPANSION PARSING	197
7.1	OVERVIEW	198
7.1.1	Similarities to Chart Parsing Methods.....	198
7.2	DEVELOPMENT	199
7.2.1	Multipass Method.....	199
7.2.2	Single Pass Method	201
7.2.3	Optimised Method.....	204
8	IMPLEMENTATION	207
8.1	OVERVIEW	208
8.2	MACRO DEFINITION TRANSLATION	209
8.2.1	Basic Translation.....	209
8.2.2	Name Mangling.....	210
8.2.2.1	Terminals and Non-terminals	210
8.2.2.2	Symbols.....	212
8.2.2.3	Precedence.....	213
8.2.2.4	Associativity.....	213
8.2.2.5	Delayed Macros.....	213
8.2.2.6	Mangling Grammar and Algorithm.....	213
8.3	IMPORT MECHANISM	215
8.4	TOKENISER	216
8.4.1	String and Character Literals.....	216
8.4.2	Multi-character Symbols	217
8.5	PARSER.....	218
8.5.1	Sub-type Non-terminal Matching.....	218
8.5.2	Partial Match Tree.....	218
8.5.3	Abstract Syntax Tree	219
8.5.4	Error Handling.....	219
8.5.4.1	Syntax Errors.....	219
8.5.4.2	Exception Errors.....	220
8.6	STANDARD USAGE.....	221
8.6.1	Command-line Arguments	221
8.6.1.1	Classpath	221
8.6.1.2	Default Imports	221

8.6.1.3	<i>Production of Java Source Code</i>	222
8.7	STANDARD LIBRARY EXTENSIONS	223
8.7.1	Quasi-quotation	223
8.7.1.1	<i>Hygiene</i>	224
8.7.1.2	<i>Unquoting</i>	225
8.7.2	Macro Definition Shorthands	226
8.7.2.1	<i>Automatic Construction Macros</i>	226
8.7.2.2	<i>Automatic Lists</i>	226
8.7.2.3	<i>Optional Macro Parameters</i>	227
8.7.2.4	<i>Statically Type-checked Parameters</i>	229
9	ANALYSIS AND COMPARISON	231
9.1	OVERVIEW	232
9.2	IMPLEMENTATION OF TEST CASES	233
9.2.1	Assertions	233
9.2.1.1	<i>Basic Implementation</i>	233
9.2.1.2	<i>Quasi-quote Implementation</i>	234
9.2.1.3	<i>Implementation Issues</i>	234
9.2.2	Iteration	234
9.2.2.1	<i>Basic Implementation</i>	234
9.2.2.2	<i>Quasi-quote Implementation</i>	235
9.2.2.3	<i>Hygienic Implementation</i>	236
9.2.2.4	<i>Static-type Matching Implementation</i>	236
9.2.3	Type-safe Formatted Output	236
9.2.3.1	<i>Example Expansion</i>	239
9.2.4	SQL Subset	239
9.2.5	Generators	241
9.2.5.1	<i>Translation of Formal Parameters</i>	243
9.2.5.2	<i>Translation of Local Variable Declarations</i>	243
9.2.5.3	<i>Generation of Resumable Code</i>	244
9.2.5.4	<i>Example Expansions</i>	244
9.2.5.5	<i>Explicit Use of GeneratorBase</i>	246
9.2.5.6	<i>Implementation Issues</i>	247
9.2.6	Haskell Subset	248
9.2.6.1	<i>Construction</i>	248
9.2.6.2	<i>Type Abstract Syntax Classes</i>	249
9.2.6.3	<i>Evaluation</i>	251
9.2.6.4	<i>Type Checking</i>	253
9.2.6.5	<i>Embedded Usage</i>	253
9.2.6.6	<i>Standalone Usage</i>	254
9.2.6.7	<i>Extended Forms</i>	255
9.2.7	Implementation Review	257
9.3	QUALITATIVE ASSESSMENT	261
9.3.1	Power	261
9.3.1.1	<i>Benchmark Test Cases</i>	262
9.3.1.2	<i>Quasi-quotation Implementation</i>	262
9.3.1.3	<i>Other Standard Library Macros</i>	262
9.3.2	Usability	263
9.3.3	Error Handling	264
9.3.4	Extensibility Criteria Assessment	264
9.4	MAYA COMPARISON	269
9.4.1	Benchmark Test Cases Comparison	269
9.4.1.1	<i>Lines of Code Comparison</i>	271
9.4.2	MultiJava	272
9.4.3	Extensibility Criteria Comparison Summary	273
9.5	GRAPH EXPANSION PARSING	277
9.5.1	Acceptable Grammars	277
9.5.2	Efficiency	277
9.5.2.1	<i>Theoretical Performance</i>	277
9.5.2.2	<i>Empirical Results</i>	278

10 CONCLUSION AND FUTURE WORK	283
10.1 CONCLUSION	284
10.2 FUTURE WORK	286
10.2.1 Flexible Lexical Analysis	286
10.2.2 Delayed Macros	287
10.2.3 Zero Argument Macros	287
10.2.4 Migration to Java 1.5	288
10.2.5 Parser Efficiency	288
10.2.6 Context-sensitive Graph Expansion Parsing	289
10.2.7 Integration of Genesis Parsing and Java Compiling	289
10.2.8 Improved Error Tracking	289
10.2.9 Usability Surveys	290
10.2.10 Library Support	290
10.2.11 Improved Embedded Haskell	290
10.2.12 Ultimate Aim	291
REFERENCES	293
A GENESIS ABSTRACT SYNTAX	305
A.1 ABSTRACT SYNTAX CLASSES	306
B GENESIS AND MAYA SIMPLE TEST CASES	313
B.1 ASSERTIONS	314
B.2 ITERATION	315
B.3 TYPE-SAFE FORMATTED OUTPUT	316

List of Code Examples

Code Example 2.1: Java Threading	21
Code Example 2.2: Ada Threading (modified from [Bar91§14.4, pp. 291]).....	22
Code Example 2.3: Java Embedded SQL	23
Code Example 2.4: Pro*C Embedded SQL	24
Code Example 2.5: C++ Macro Misuse.....	26
Code Example 2.6: C++ Macro Parsing Difficulties	26
Code Example 2.7: Powerful C++ Macro Construct	27
Code Example 2.8: C++ Assertion Macro.....	27
Code Example 2.9: C++ Iteration Macro.....	28
Code Example 2.10: Icon Fibonacci Generator	29
Code Example 2.11: Icon Fibonacci Sequence.....	29
Code Example 2.12: Improved Icon Fibonacci Sequence	29
Code Example 2.13: C++ Generator Helper Classes	30
Code Example 2.14: Basic C++ Generator Macros	31
Code Example 2.15: C++ Fibonacci Generator	31
Code Example 2.16: C++ Fibonacci Generator After Macro Expansion	32
Code Example 2.17: Extended C++ Generator Macros	33
Code Example 2.18: Improved C++ Fibonacci Generator	33
Code Example 2.19: C++ Message Map Macro Usage	34
Code Example 2.20: C++ Debugging Macro	34
Code Example 2.21: C++ Debugging Macro Usage.....	35
Code Example 2.22: C++ Debugging Macro After Expansion	35
Code Example 2.23: C++ Template Meta-programming	36
Code Example 3.1: Assertions.....	61
Code Example 3.2: Java1.5 Assertions.....	62
Code Example 3.3: Maya Assertions.....	63
Code Example 3.4: Iteration	63
Code Example 3.5: C++ STL Iteration	64
Code Example 3.6: Java1.5 Iteration	64
Code Example 3.7: Maya Iteration	65
Code Example 3.8: Typesafe Formatted Output.....	65
Code Example 3.9: Java1.5 Typesafe Formatted Output.....	67
Code Example 3.10: Embedded SQL Subset.....	68
Code Example 3.11: Java Generators	69
Code Example 3.12: Embedded Haskell Subset.....	70
Code Example 3.13: map Function	70
Code Example 3.14: map Function with Type Signature.....	71
Code Example 3.15: map Function Without Pattern Matching.....	72
Code Example 3.16 map Function Using Lambda Functions	72
Code Example 3.17: List Comprehension Decomposition	73
Code Example 4.1: Simple Lisp Program Fragment	79
Code Example 4.2: Lisp Quotation Expression for Code Example 4.1	79
Code Example 4.3: Lisp Simulated Infix Expressions.....	80
Code Example 4.4: defmacro	80
Code Example 4.5: define-syntax.....	81
Code Example 4.6: define-syntax General Form	81
Code Example 4.7: Quasi-quote and Unquote.....	82
Code Example 4.8: swap Function.....	82
Code Example 4.9: Improved swap Function	83
Code Example 4.10: Hygienic swap Function	83
Code Example 4.11: swap Function using syntax-case	84
Code Example 4.12: swap Function with Error Handling.....	85
Code Example 4.13: Template Haskell printf Expansion	90
Code Example 4.14: Template Haskell Assertions.....	91

Code Example 4.15: Illegal Template Haskell Quasi-quotation	91
Code Example 4.16: Template Haskell <code>printf</code> Definition	92
Code Example 4.17: Template Haskell <code>printf</code> Usage	93
Code Example 4.18: Template Haskell N-tuple Selection	94
Code Example 4.19: Template N-tuple Selection Expansion	94
Code Example 4.20: MS^2 Enumerations	97
Code Example 4.21: MS^2 Dynamic Binding	99
Code Example 4.22: Printable Enumerations	101
Code Example 4.23: JSE Iteration Definition	109
Code Example 4.24: JSE Improved Iteration Definition	109
Code Example 4.25: Open Java Visitor Methods Usage	113
Code Example 4.26: Open Java Visitor Methods Definition	114
Code Example 4.27: Maya Assertions	120
Code Example 4.28: Maya Iteration Usage and Expansion	121
Code Example 4.29: Partial Maya Iteration Definition	122
Code Example 5.1: Grammar to Macro Translation	133
Code Example 5.2: Raw and Quoted Terminals	134
Code Example 5.3: Precedence Syntax	134
Code Example 5.4: Associativity Syntax	136
Code Example 5.5: <code>printf</code> Macro Prototypes	147
Code Example 5.6: Nested Macro Use	148
Code Example 5.7: Ambiguous Java Declarations	150
Code Example 5.8: Macro Reflection Methods	158
Code Example 6.1: Simple Expression Predictive Parser	173
Code Example 6.2: Table-driven Simple Expression Parser	175
Code Example 6.3: Simple Expression Operator-precedence Parser	181
Code Example 6.4: Simple Expression LR Parser	184
Code Example 6.5: CYK Algorithm	190
Code Example 6.6: Earley's Algorithm	192
Code Example 7.1: Multipass Graph Expansion Algorithm	199
Code Example 7.2: Single Pass Graph Expansion Algorithm	201
Code Example 7.3: Final Graph Expansion Algorithm	204
Code Example 8.1: Basic Macro Translation	209
Code Example 8.2: Basic Name Mangling	210
Code Example 8.3: Java Reserved Word Mangling	211
Code Example 8.4: Mangled Name Conflict Resolution	212
Code Example 8.5: Symbol Mangling	212
Code Example 8.6: Code Fragment Without Multi-character Symbols	216
Code Example 8.7: Code Fragment Containing Multi-character Symbols	217
Code Example 8.8: Genesis Quasi-quotation	223
Code Example 8.9: Partial Basic Quasi-quotation Definition	224
Code Example 8.10: Unquoting Implementation	225
Code Example 8.11: Optional Macro Parameter Class	228
Code Example 8.12: Extended Macro Parameter List	228
Code Example 8.13: Implementation Outline of Macro Definitions With Optional Arguments	229
Code Example 8.14: Factorial Literal Specialisation	229
Code Example 9.1: Basic Assertion Implementation	233
Code Example 9.2: Quasi-quote Assertion Implementation	234
Code Example 9.3: Basic Iteration Implementation	235
Code Example 9.4: Quasi-quote Iteration Implementation	235
Code Example 9.5: Hygienic Iteration Implementation	236
Code Example 9.6: Static-Type Matching Iteration Implementation	236
Code Example 9.7: Partial Type-safe Formatted Output Implementation	238
Code Example 9.8: Type-safe Formatted Output Expansion	239
Code Example 9.9: Partial SQL Subset Implementation	240
Code Example 9.10: Generator Helper Class	241
Code Example 9.11: Suspend Statement Implementation	241

Code Example 9.12: Generator Method Implementation	242
Code Example 9.13: Translation of Formal Parameters	243
Code Example 9.14: Translation of Local Variable Declarations	243
Code Example 9.15: Repeating Generator Expansion	244
Code Example 9.16: Fibonacci Generator Expansion	246
Code Example 9.17: Sub-sequence Generator Expansion and Use	247
Code Example 9.18: FunObject Interface	248
Code Example 9.19: Haskell Subset Cons Abstract Syntax Class	249
Code Example 9.20: Type Abstract Syntax Classes for the Haskell Subset	250
Code Example 9.21: Evaluation of Function Application	252
Code Example 9.22: Haskell Subset if Expression	253
Code Example 9.23: Embedded Haskell Wrapper Definition	254
Code Example 9.24: Standalone Haskell Module	254
Code Example 9.25: Standalone Haskell main Method	255
Code Example 9.26: Function Declarations Definition	256
Code Example 9.27: Operator Currying Definition	256
Code Example 9.28: Simple Single Source List Comprehensions	256
Code Example 9.29: Multiple Condition Single Source List Comprehensions	257
Code Example 10.1: Extended Tokeniser Possibility	286
Code Example 10.2: Macro Expansion Requiring Delayed Macros	287
Code Example A.1: Compilation Unit Classes	304
Code Example A.2: Type Classes	305
Code Example A.3: Statement Classes	305
Code Example A.4: Variable Declaration Classes	306
Code Example A.5: Method and Type Declaration Classes	307
Code Example A.6: Expression Classes	308
Code Example A.7: Identifier Classes	309
Code Example A.8: Literal Classes	309
Code Example B.1: Genesis Assertion Definition	314
Code Example B.2: Maya Assertion Definition	314
Code Example B.3: Genesis Iterator Definition	315
Code Example B.4: Maya Iterator Definition	315
Code Example B.5: Genesis Type-safe Formatted Output Definition	316
Code Example B.6: Maya Type-safe Formatted Output Definition	316

List of Figures

Figure 3.1: SQL Subset Grammar	68
Figure 3.2: Haskell Subset Grammar	73
Figure 3.3: Haskell Embedding Grammar	74
Figure 4.1: Lisp S-expression for Code Example 4.1	79
Figure 4.2: JSE Call and Statement Macro Grammars.....	106
Figure 5.1: Expression Grammar	135
Figure 5.2: Method Definition Grammar	137
Figure 5.3: Genesis Grammar	139
Figure 5.4: Simple Tokenising	141
Figure 5.5: Multi-character Symbol Grouping Tokenising	141
Figure 5.6: Single-character Symbol Output Tokenising	142
Figure 5.7: Multi-character Symbol Combinations Tokenising	143
Figure 5.8: Tokeniser Transition Diagram	145
Figure 5.9: Same Sub-rule Precedence.....	151
Figure 5.10: Different Sub-rule Precedence	152
Figure 5.11: Associativity	152
Figure 5.12: Partial Abstract Syntax Type Hierarchy	156
Figure 5.13: Exception Type Hierarchy	157
Figure 6.1: Generalised Grammar Rule	163
Figure 6.2: BNF Context-free Grammar	165
Figure 6.3: Ambiguous Simple Expression Grammar.....	166
Figure 6.4: Unambiguous Simple Expression Grammar	166
Figure 6.5: Non Left-recursive Simple Expression Grammar	167
Figure 6.6: High-level Compiler Model	168
Figure 6.7: Derivations.....	169
Figure 6.8: Non-ambiguous Unparsable Grammars.....	170
Figure 6.9: Deterministic Finite Automaton	171
Figure 6.10: Table-driven Predictive Parser Model	174
Figure 6.11: Shift-reduce Parser Reductions.....	178
Figure 6.12: Shift-reduce Parser Model	179
Figure 6.13: Operator-precedence Parser Model.....	180
Figure 6.14: LR Parser Model	182
Figure 6.15: Expression Grammar in Chomsky Normal Form	191
Figure 6.16: CYK Expression Parsing Recognition Table	191
Figure 6.17: Chart Parser Representation.....	193
Figure 7.1: Multipass Graph Expansion	200
Figure 7.2: Single Pass Graph Expansion	202
Figure 7.3: Simple Expression Partial Match Tree	205
Figure 7.4: Dangling-else Partial Match Tree	205
Figure 8.1: Genesis Compiler Structure	208
Figure 8.2: Precedence Mangling Examples.....	213
Figure 8.3: Mangled Name Grammar	214
Figure 8.4: Name Mangling Algorithm.....	214
Figure 8.5: Graph Produced From Tokenising Code Example 8.6	216
Figure 8.6: Graph Produced From Tokenising Code Example 8.7	217
Figure 8.7: Method Declaration Fragment	227
Figure 9.1: Differing Time Complexities of Earley's Algorithm	278
Figure A.1: High-level Abstract Syntax Class Hierarchy	306
Figure A.2: Declaration Abstract Syntax Class Hierarchy	306
Figure A.3: Expression Abstract Syntax Class Hierarchy.....	308

List of Tables

Table 3.1: Criteria for rating an Extensible Language's Power	56
Table 3.2: Criteria for rating an Extensible Language's Usability	57
Table 3.3: Criteria for rating an Extensible Language's Error Handling.....	58
Table 3.4: Benchmark Test Cases Summary	60
Table 4.1: Lisp Applicability to Benchmark Test Suite	85
Table 4.2: Lisp Extensibility Criteria Assessment.....	86
Table 4.3: Template Haskell Applicability to Benchmark Test Suite.....	95
Table 4.4: Template Haskell Extensibility Criteria Assessment.....	95
Table 4.5: MS ² Applicability to Benchmark Test Suite.....	102
Table 4.6: MS ² Extensibility Criteria Assessment.....	103
Table 4.7: JSE Applicability to Benchmark Test Suite.....	110
Table 4.8: JSE Extensibility Criteria Assessment.....	111
Table 4.9: OpenJava Applicability to Benchmark Test Suite	115
Table 4.10: OpenJava Extensibility Criteria Assessment	115
Table 4.11: Maya Applicability to Benchmark Test Suite.....	123
Table 4.12: Maya Extensibility Criteria Assessment.....	124
Table 6.1: Chomsky Hierarchy.....	163
Table 6.2: Simple Expression LL Parse Table.....	176
Table 6.3: Simple Expression Operator-precedence Table.....	182
Table 6.4: Simple Expression LR Parse Table	185
Table 6.5: Simple Expression LALR Parse Table	186
Table 7.1: Single Pass Graph Expansion Evaluation.....	203
Table 7.2: Final Algorithm Graph Expansion Evaluation	205
Table 9.1: Rules for Function Type Equality.....	250
Table 9.2: Genesis Applicability to Benchmark Test Suite	257
Table 9.3: Genesis Extensibility Criteria Assessment	264
Table 9.4: Genesis and Maya Benchmark Test Suite Comparison	269
Table 9.5: Genesis and Maya Lines of Code Comparison.....	271
Table 9.6: Genesis and Maya Extensibility Criteria Comparison.....	273
Table 9.7: Earley Versus GEP Time Complexity	279
Table 9.8: Earley Versus GEP Comparison.....	281